CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# APIs as Digital Innovation Objects – Implications on Requirements Elicitation

Bachelor of Science Thesis in Software Engineering and Management

Mikaela Törnlund
Mohammad Ahraz Asif

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



**Presenting and analysing a method for requirement elicitation for strategic APIs**

Mikaela Törnlund
Mohammad Ahraz Asif



Supervisor: Eric Knauss
Examiner: Jennifer Horkoff

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

# APIs as Digital Innovation Objects - Implications on Requirements Elicitation

Mikaela Törnlund
Software Engineering and Management BSc
Gothenburg University
Gothenburg, Sweden
tornlundmikaela@gmail.com

Mohammad Ahraz Asif
Software Engineering and Management BSc
Gothenburg University
Gothenburg, Sweden
ahraz.asif1994@gmail.com

***Abstract*—APIs as a way to expose business assets have become common across industry. Consequently insights into strategic API development practices are becoming more and more relevant. However domain specific knowledge for making strategic design decisions for APIs is often lacking. In this study we present and analyse a method for requirement elicitation for APIs created through use of the design science methodology alongside careful consideration of standard requirements elicitation practices and API strategies - specifically insights from the 'API as Digital Innovation Objects' model on API strategy. We consider this due to a lack of awareness about what information is significant for such strategic design decisions, especially during requirements elicitation. Our findings show that there are several aspects of strategic API design that can be considered in order to improve requirements elicitation. The results imply that strategic API considerations change requirements elicitation by changing how it is carried out - not necessarily the tasks and activities conducted.**

***Keywords*-API, API strategy, requirements elicitation, Digital Innovation Object;**

## I. Introduction

Application Programming Interfaces (commonly referred to as APIs) are interfaces designed to expose part(s) of, or wholly expose business asset(s) controlled by an organization so that others parties with an interest in that asset may utilize it. These parties may range from internal departments to third party developers developing a separate service. The usage & benefits of APIs are well documented [1]- such as increasing development speed [2], and they are more and more seen as products of a business that need continuous enhancement [3] during their lifecycle.

APIs are often considered normal 'software artefacts', of which the development effort is often approached using generalized practices. However, as presented in the 'Software Development Day' [4] presentation held by Hammouda et al on API development, APIs have a specific subset of issues and nuances therefore it is important to develop and understand approaches that are tailored for API design and development. They suggested that there are some conceptual frameworks regarding APIs that exist such as considering APIs as 'Digital Innovation Objects'. However, these are emergent concepts that are not yet fully established.

Consideration of APIs as 'Digital Innovation Objects' is a conceptual model for reasoning about API strategy. Due to how decoupled APIs are from other systems and their implied layered architecture they exhibit digital object properties [5] such as interactivity and editability. Hammouda et al motivate that this loosely coupled layered architecture approach causes the emergence of four layers relevant for strategic API reasoning.

It was our intent to build upon current solutions regarding requirements elicitation by leveraging API strategy considerations through the use of the 'APIs as Digital Innovation Objects' framework. In this study, using the design science research methodology [6] we present and analyse an API specific method for eliciting requirements. We define a requirements elicitation method to be a collection of tools, techniques and practices aggregated for use in a specific development domain. With our study we aimed to propose an initial method for requirements elicitation for APIs and investigate the relevance of API strategy towards requirements elicitation.

During our study, SoftwareSkills AB, a small recruitment agency and skill testing company specializing in the software development industry intended on developing an internal API for one of their core modules which was being transferred to a micro-service. This presented a valuable opportunity conducting research regarding API strategy during their API development.

We participated in the development of their micro-service API to obtain a better understanding of API requirements engineering and how the 'Digital Innovation Object' framework can help drive requirements elicitation. The output requirement elicitation method proposed was evaluated through a survey by several industry API experts and students with relevant experience as part of an expert evaluation. Through our evaluation we were able to determine the validity of the requirements elicitation method produced and the coverage of API specific needs.

## II. Background

Digital objects differ from physical items on several levels. As explained by Kallinikos et al [5], digital objects have an ambivalent ontology (i.e. hard to define) but can be distinguished from physical objects by their specific attributes. There are two main attributes, the first of them being modularity which is explained as the blocks of functionality that make

up a system. They are relatively independent with a loosely coupled relationship between them. Second being granularity, which is defined as the collection of items and subroutines of which the blocks contain. From these two attributes, others can be derived to further specify what constitutes a digital object. Kallinikos et al state these derived attributes to be:

1) Editability, which is defined as the ability to change and update the content of the digital object and the items within the object .
2) Interactivity, which is defined as enabling plausible actions depending on the choice of the user.
3) Openess, which is explained as the ability to access and modify the digital objects through the use of other digital objects. Therefore controlling the behavior of the object.
4) Distributedness, which is explained in regards to digital objects being borderless and unable to be bound to distinct entities as they are rarely enclosed to a single source.

Kallinikos et al also note the beneficial capacity of digital objects alongside the problematic manageability that follows their constantly fluctuating relationships.

As seen in the works of de Souza and Redmiles [1], APIs provide a unique benefit when it comes to being able to expose only sections of business assets, thereby maintaining the principle of 'information hiding'. APIs allow for modularization of the over-arching software system and the modularization of the tasks associated with its development. Aitamurto and Lewis [3] motivate the widespread adoption of APIs throughout industries by providing examples of news organizations which use APIs to provide access to content (business assets).

In the 'Software Development Day' presentation about API development [4] the authors describe how APIs can be viewed as 'Digital Innovation Objects', and how their modularity and granularity lead to an implied layered architecture and what the implications are of considering APIs through this lens. They expose four specific layers with their results, as seen in the snapshot of their presentation in figure 1: a domain layer, app SW layer, API layer and business asset layer. They define the layers as such:

*a) Domain:* The domain layer encapsulates needs or events that are satisfied by application software implemented on top of the API. An example of this would be use-cases concerning voice-recognition based home assistant technologies. Concerns regarding the domain layer could be: What are the anticipated changes within the domain? What are the business models within the domain?

*b) App SW:* The application software layer encapsulates software systems that directly consume one or several APIs for the purposes of meeting the domain needs. Following on from the previous example, an example of application software could be a voice-recognition application consuming natural language processing APIs. Concerns regarding the application software layer could be: Is it being developed internally or externally? Does the app SW provide additional functionality?

*c) API:* The API layer encapsulates the entity which enables access to one or more business assets. For our example this would be the API which exposes features from the natural language processing algorithm used. Concerns regarding the API layer could be: How much control over the business asset should be afforded to the API consumers (i.e. chatty vs chunky endpoints)? What methods of authentication will be used?

*d) Business Asset:* The business asset layer covers interests regarding any property of the business asset such as the the implementation or the data exposed. For our example this would encapsulate concerns regarding the natural language processing algorithm itself. Concerns regarding the business asset could be: What features of the business asset do we want to expose? Are there new business assets that need to be exposed?

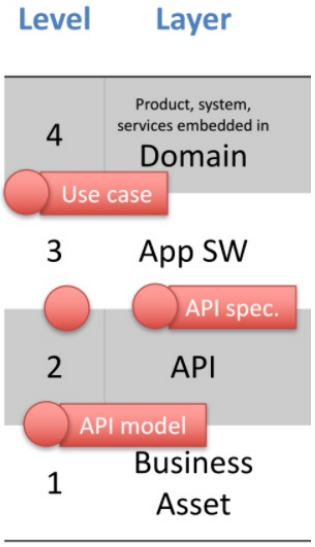



Fig. 1. Emergent layers from the 'APIs as Digital Innovation Objects' model. Taken from the presentation by Hammouda et al [4]

All layers are influenced directly by the layers they are bound to. The layered approach leads to the definition of three possible boundary objects: use cases (between the domain & app SW layer), the API specification (between the app SW & API layer) and the API model (between the API layer & business asset layer). These boundary objects are influenced by, and directly influence the adjacent layers.

Zowghi and Coulin [7] state in their exploratory analysis of requirements elicitation that it is made up of the following activities, which are to be conducted by a requirements engineer:

1) Understanding the application domain
   a) Examination of the real world environment in which the system will be utilized.
   b) Consideration of political, organizational and social aspects of the project.
   c) Consideration of the problems that need to be solved in relation to the business goals and issues of the project

2) Identifying the source of requirements
   a) Requirements may be spread across many sources and exist in a variety of formats.
   b) Possible sources include: stakeholders, users, subject matter experts, existing systems and documentation.
3) Analyzing the stakeholders
   a) Creation of a list of any potential parties who many have an interest in the system.
   b) Prioritization of all interested parties in regards to whose interests are the most valuable
4) Selecting tools, techniques and approaches
   a) Making the appropriate choices in regards to exactly what techniques, tools and approaches the requirements engineer will utilize
   b) Choosing a variety of techniques and approaches to achieve optimal results given the context of the software project.
5) Eliciting requirements
   a) After execution of all preceding activities, the elicitation of requirements begins.
   b) Establishing the scope of the project.
   c) Investigate needs and wants of stakeholders.
   d) Determining future processes of the current system.

The requirements elicitation is highly dependent within the context it is performed. Zowghi and Coulin motivate that organizational maturity and size all have an impact on the process and the output (the requirements) alongside the volatility of the project (the measure of how frequently requirements change during software development). They state the procedure is to be carried out by a requirements engineer, whose responsibilities are defined as:

1) Exploring the problem domain
   a) Obtain an understanding of the problem domain which needs to be solved
2) Managing and communicating the elicitation process to stakeholders
   a) Plan, execute and communicate requirements elicitation meetings between the stakeholders
3) Facilitate relevant conversation and involve all parties to the intent of satisfying their participation requirement
   a) Ensure all stakeholders feel they are contributing to the requirements and maintain as much coverage of stakeholder satisfaction as possible
4) Mediate conflicts between elicited requirements and stakeholders
   a) Conflicts may arise at several points in the requirements elicitation, such as when prioritizing requirements in regards to the source stakeholder
5) Documenting the requirements produced
   a) As a record and the medium through which the requirements may be validated.
6) Fulfill missing roles
   a) Assuming the relevant 'developer community' role when such a stakeholder has not been assigned. For example a requirements engineer may have to consider the point of view of a tester when making decisions that will impact the role in the future.
7) Validating the requirements
   a) Through external validation - cross-referencing the requirements against external documentation
   b) Through internal validation - cross-referencing the requirements against internal documentation

Zowghi and Coulin define 19 different techniques and approaches available to requirements engineers to be utilized during the requirements elicitation. While there are over hundreds of techniques and approaches to choose from, they motivate that the following core group of 8 are the most representative of the ones currently most mentioned in modern literature:

1) Interviews
2) Domain analysis
3) Group Work
4) Ethnography
5) Prototyping
6) Goals
7) Scenarios
8) Viewpoints

There are a subset of requirements can be characterised as 'architecturally significant requirements'. These are requirements that have a "measurable impact on a software system's architecture" and have a "high cost associated to change" as defined by Chen et al in their characterisation of architecturally significant requirements [8]. As a consequence of these characteristics if they are incorrect or overlooked an architecture informed by those requirements will likely be flawed. There are several characteristics of architecturally significant requirements:

*a) Wide Impact:* Architecturally significant requirements typically have a big impact on the system.

*b) Target trade-offs:* The consideration of trade-offs (i.e. making a compromise when all involved requirements cannot be satisfied) for a requirement indicate that it is architecturally significant.

*c) Strictness:* When a requirement imposes a design option due to its non-negotiable nature it is typically an architecturally significant requirement.

*d) Assumption breaking:* When a requirement breaks pre-existing assumptions regarding a systems architecture (i.e. a service needs to become available 24/7 due to expansion into foreign markets) this indicates that it is an architecturally significant requirement.

*e) Difficult to achieve:* In case a requirement is difficult to meet or requires complex technical solutions it is likely to be architecturally significant.

In order to identify these characteristics a requirements engineer typically needs a solid foundation of knowledge within the solution space, as defined by Chen et al. They motivate

that in order to identify the characteristics of architecturally significant requirements where a requirements engineer might not have solution space expertise a set of heuristics can be utilised. These heuristics can guide the requirements engineer towards questions and concerns that are architecturally significant:

*a) Quality attributes:* If a requirement specifies a quality attribute for a given system it implies that the requirement is architecturally significant.

*b) Core features:* A requirement regarding the core functionality of a system (i.e. the problems the system is trying to solve) is typically architecturally significant.

*c) Constraints:* Requirements that impose constraints on the project due to organizational, process, technical or financial reasons are usually architecturally significant.

*d) Application Environment:* In case a requirement specifies the environment the application is to be run (i.e. the Internet, virtual machines, mobile devices, etc) it is indicative of a architecturally significant requirement.

Due to the emergent nature of recent API strategy insights, the body of work regarding systemic approaches and research about appropriate practices and difficulties within API design, development and management are lacking. Fortunately however, a large body of work exists in regards to requirements elicitation as a general practice in software development. By leveraging this large body work and the emerging insights into API strategy we hope to address the lack of literature regarding API design, development and management - specifically requirements elicitation informed by API strategy.

## III. Research questions

Through reviewing standard requirements elicitation practices and then conducting a participant observation during active strategic API development within industry, we intended on learning about API specific needs in the area of requirements elicitation. This allowed us to investigate our first research question:

- R.Q.1: What is specific to requirements elicitation for strategic design of APIs?

The results of the participant observation and literature analysis and the integration of those findings into the requirements elicitation method were the basis of investigating R.Q.1, as they informed us of deviations from the standard requirements elicitation approaches when it comes to requirements elicitation for strategic APIs.

During our investigation, we integrated API strategy considerations into our requirements elicitation method by reasoning about API strategy through the use of the 'APIs as Digital Innovation Objects' conceptual framework. This enabled us to investigate our second research question:

- R.Q.2: How can insights from considering 'APIs as Digital Innovation Objects' improve requirement elicitation for APIs?

Through the previous investigations regarding how 'APIs as Digital Innovation Objects' improve requirements elicitation and what was specific towards requirements elicitation for strategic API design, we were able to investigate our final research question:

- R.Q.3: To what extent is it feasible to consider API strategies for driving requirements elicitation?

By integrating API strategy concerns into the requirements elicitation method, we were able to identify the areas of requirements elicitation that were impacted and to what degree. In turn this provided us with the results to reason about the feasibility of considering API strategies for driving requirements elicitation.

## IV. Methodology

Our research can be classified as a design science research [6], which is a solution generating model. Design science research is conducted through the definition of ideas, practices, technical capabilities or products (i.e. a design artefact) and their analysis. The process utilised is an iterative cycle named the 'regulative cycle', with five phases [9] [10] [11]: problem investigation, design candidate, validation, implementation and evaluation. After conducting these five phases, the cycle continues using the output from the previous iteration as the input for the next.

### A. Regulative Cycle

*1) Problem Investigation:* The first phase of the regulative cycle is problem investigation. The purpose of this phase is to obtain or deepen knowledge about the problem domain. This phase generally consists of literature reviews, interviews or other exploratory measures in order to investigate the problem. In regards to our study, the problem domain is requirements elicitation for strategic API development and emerging insights into considering API strategy. The aim of this phase in all our iterations was to obtain a deeper understanding of 'APIs as Digital Innovation Objects', requirements elicitation and other relevant areas that emerge.

*2) Design Candidate:* The second phase of the regulative cycle is presenting a design candidate. This phase involved finding potential ways of solving the problem identified in the previous step. Multiple solutions can be generated, however there is no guarantee that any one solution will be implemented. Similar to the previous step, this is an exploratory phase where all options are explored. For our study, this step involved generating a abstract representation of the objectives of our requirements elicitation method for validation in the next step.

*3) Validation:* The third phase of the regulative cycle is validating the different design candidates generated. The purpose of this phase is to evaluate the design candidate in regards to the problem criteria defined in the first step, and how well the design candidate meets those goals. This can be done through various means, however for our study this involved a review of the design candidate by relevant experts in the field and a literature review regarding coverage of standard requirements elicitation practices by the requirements elicitation method.

*4) Implementation:* The fourth phase of the regulative cycle involves implementing the validated design candidate from the previous step. Typically this step can be conducted in many different ways, and is very dependant on the complexity and domain of the solution being created. In regards to our study this step involved satisfying the abstract goals of the different steps of the requirements elicitation method presented in step two with specific tasks to meet those objectives.

*5) Evaluation:* The final phase of the regulative cycle is evaluation of the implemented design candidate. The purpose of this phase is to determine if the implemented solution solves the problem identified. Through this evaluation, not only is the design candidate analysed but typically a deeper understanding the the problem area is obtained which acts as the input for the first step of the next iteration of the regulative cycle. In regards to our study, evaluation of the implementation was done through review by relevant experts and a survey which was circulated to industrial practitioners alongside students with relevant experience.

### *B. Data Collection Methods*

Data collected in this study was both quantitative and qualitative.

*1) Participant Observation:* For our first iteration of the regulative cycle, the primary method of data collection was conducting participant observation at the company SoftwareSkills AB during the development of an internal API to expose a business asset in the form of a black box code testing service. Named the 'Evaluator', it was designed to evaluate submitted algorithms from several different languages.

Participant Observation, as defined by Lethbridge et al [12] can be seen as the researcher essentially becoming a 'part of the team'. This involves the researcher participating in the entire development process as well as observing the other actors present. This was chosen as often much work conducted by developers occurs 'in their head', as stated by Seaman [13] in her breakdown of qualitative methods in empirical software engineering research. By being active participants in the production of a real, industrial API we were able to gain insights into strategic API development that would have otherwise been very difficult to obtain.

*a) Case Company Description:* The duration of the participant observation at SoftwareSkills lasted eight weeks. The company employed use of an agile software development process, SCRUM. The sprints lasted one week, and after each sprint there was a sprint retrospective and sprint planning meeting. Additionally there were daily morning stand-ups conducted by the development team. Throughout the entirety of the participant observation, we were participating in development of both the API, and the system consuming the API. The intent of the API was to expose the unique business asset of SoftwareSkills, the 'Evaluator'. This evaluator acts as a black box code testing service for candidates applying to jobs. The purpose of the evaluator is to accept algorithm submissions in five different programming languages (Csharp, C++, Java, JavaScript & Python), execute them in a sandboxed testing environment where they receive different inputs and then evaluate the outputs and the time taken for the submission to run. Previous to our development, the Evaluator was highly coupled to a separate application server hosted by SoftwareSkills. The main application server would act as a gateway to the secondary application server, each with their separate clients and business logic. The intent making this API was to consolidate their two different application servers into one service.

*2) Interviews:* In order to conduct the validation of our initial design candidate, the subsequent evaluation of the first iteration of the regulative cycle and the problem investigation for the second iteration of the regulative cycle we employed the use of interviews with several experts. We identified experts as researchers who have experience in the fields relevant to our design candidates, i.e. requirements engineering, 'APIs as Digital Innovation Objects' and API strategy. This can be classified as the use of 'expert evaluation' - Peffers et al [14] state in their design science research evaluation, expert evaluation is assessment of an artefact by experts with knowledge in the relevant area.

The first set of interviews conducted for the purposes of validating the initial design candidate of the first iteration was of an unstructured form with a requirements elicitation expert. During this interview, the expert was asked to evaluate and provide suggestions upon the initial findings which created the foundation of the requirements elicitation method. As this interview was at a very early stage in the study an unstructured approach was chosen as suitable as the means of evaluation of the design candidate were unclear. By having an unstructured interview, the expert was able to provide direct feedback on what they considered was lacking within the design candidate.

The subsequent interviews conducted were of a semi-structured nature. An interview guide containing the first implementation of the requirements elicitation method was sent out to three experts within the following fields: API strategy, 'APIs as Digital Innovation Objects' and Requirements Engineering. The experts were asked to read the implementation provided alongside the interview questions beforehand. The interview questions was composed of three parts: one section regarding evaluation of the requirements elicitation method, the second section regarding problem investigations for APIs and requirements engineering and finally a third section regarding problem investigations for 'APIs as Digital Innovation Objects'. All of the evaluation and problem investigation interviews were recorded. During the interview, the interview subjects were taken through the implementation of the design artefact with chances to make comments at any time. Following the walk-through, the interview subjects were then asked the questions from the interview guide. The interview guide can seen in Appendix B.

*3) Surveys:* In order to conduct the final evaluation of the design artefact we prepared a presentation and survey presenting the intent and methodology of the study conducted, a summarized version of the design artefact (as seen in table

II of section V), the dependancy visualization (as seen in figure 2) and the detailed implementation of the design artefact attached (as seen in Appendix A). Following the presentation, the participants were asked to conduct a survey evaluating: the quality of the design artefact, it's usefulness compared to standard requirements elicitation practices, coverage of API specific needs and other questions regarding the execution of the design artefact within a company and whether the method provided novel considerations regarding requirements elicitation for APIs.

Developers with experience using, developing or managing APIs from several companies (Bisnode AB, iCore Solutions AB & VisitGroup AB) were approached to participate in the survey. Additionally several student developers and hobbyists with experience using APIs were approached. The survey results can be seen in Appendix C, with the results explained in VII.B.

Based on the answers provided we were able to quantitatively and qualitatively evaluate the quality of the requirements elicitation method in regards to coverage of API specific concerns and the feasibility of considering API strategy within requirements elicitation.

## V. Requirements Elicitation Method

### A. Requirements Elicitation Method Final Design Candidate

TABLE I
Objectives of Requirements Elicitation for Strategic Design of APIs

| Process Areas | Objectives |
|---|---|
| Background Information Collection | **Identify business model** |
| | Identify stakeholders, their desires, identify their unwanted features, identify their involvement |
| | Identify project constraints: time, budget and resources |
| | Identify long-term plans from the organization that may have an impact on the project |
| | Identify the business problems caused without the presence of the API |
| | **Identify process requirements** |
| **Technical Requirement Collection** | **Determine technical constraints** |
| | **Determine parallel deployment needs** |
| Functional Requirement Collection | Determine the business asset to be exposed |
| | Determine the operations to be conducted on the business asset |
| | Determine the granularity of the API endpoints |
| Non-Functional Requirement Collection | Determine appropriate levels of Performance |
| | Determine appropriate levels of Uptime |
| | Determine appropriate levels of Security |
| | **Determine appropriate levels of Usability** |
| | Determine form of documentation |
| | Determine maintenance/update needs |
| | Determine appropriate levels of Scalability |
| | **Determine appropriate levels of testing** |

The final design candidate achieved after both iterations is presented in table I. The sections in bold represent findings from the second iteration. This is an overview of the process areas we have defined, with the goals of each process area listed.

Many of the objectives are dependent on other objectives within the method being finished - in order to easily identify these, we initially present a dependency visualization in figure two of the different objectives and then following that we present our method with the objectives broken down into tasks in detail. One important consideration to make is that while we present some form of chronological order to the elicitation activities, they are to be interpreted as loose guidelines. They are by no means concrete and will have to be shifted on a case-by-case basis. Requirements elicitation is an informal process by nature and different projects will require different orders of activities. Ideally the elicitor should employ the use of an

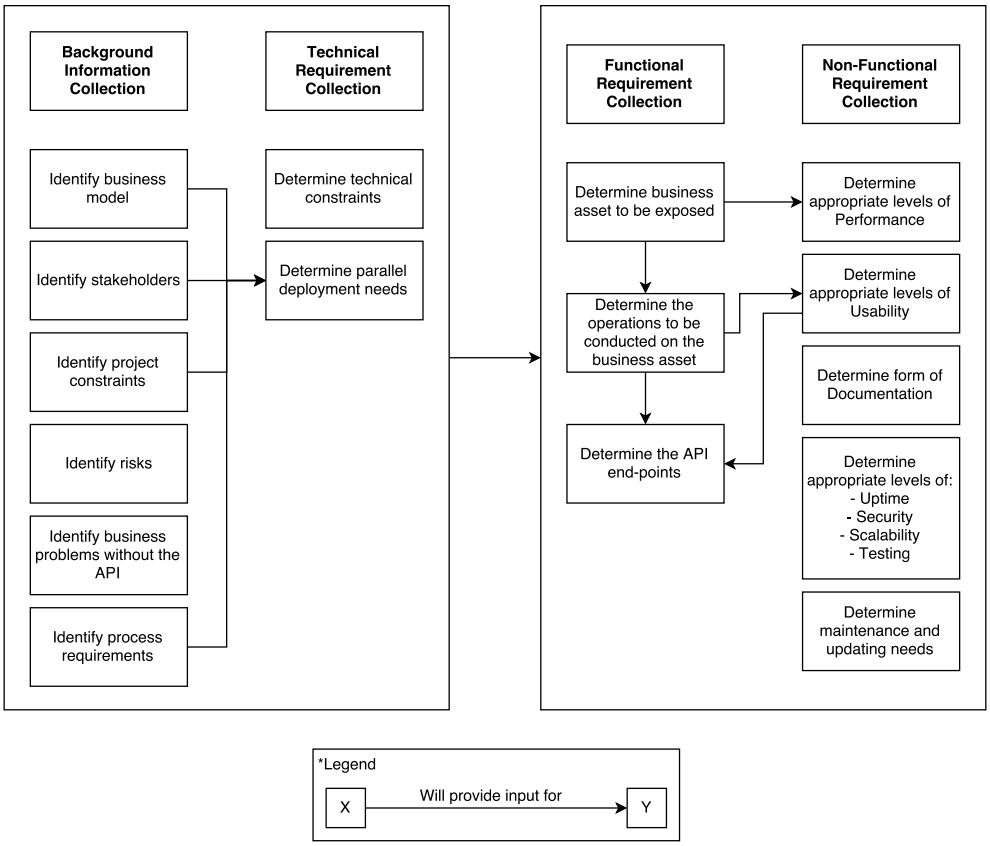



Fig. 2. Rough chronology for conducting requirements elicitation objectives

agile methodology, as many interviews will be repeated until a final set of requirements has been realized.

### *B. Requirements Elicitation Method Final Implementation*

This section presents a summarized version of the requirements elicitation method created as the design artefact of this study, as seen in table II. Sections in bold represent findings from the second iteration. The method provides four 'process areas' identified for collecting requirements for strategic API design. The process areas represent the collection of different types of information. These process areas contain several objectives which should be satisfied.

The objectives are broken down in to tasks which provide useful artefacts or information in order to elicit requirements for the API. The primary outputs of this elicitation method are the *API specification* & the *API model* as seen as the boundary objects of the API layer within the 'APIs as Digital Innovation Objects' model [4].

Additionally the subjects of the interviews suggested are merely guidelines - within organizations information is typically distributed and the elicitor will have to find valid sources for the data to be collected within the tasks. In case the guidelines provided do not direct the elicitor towards the right source of information, the technique of 'snowball sampling' [15] should be utilised. The elicitor should question the suggested subjects as to who they believe will have the information required - continuing to do so until an adequate source has been found.

The detailed version of the requirements elicitation method can be seen in Appendix A. We highly recommend reviewing this version in order to gain a better understanding of the requirments elicitation method.

TABLE II
OBJECTIVES & TASKS OF REQUIREMENTS ELICITATION FOR STRATEGIC DESIGN OF APIS

| Objective | Task | Output |
|---|---|---|
| **Background Information Collection** | | |
| **Identify business model** | **Interview senior management & product owner** | **Description of the business model surrounding the business asset(s) to be exposed** |
| Identify stakeholders | Create Personas for API consumers with product owner | Personas encapsulating typical API consumers |
| | Identify other relevant stakeholders with product owner | List of stakeholders with their information |
| | **Prioritise stakeholders with product owner and senior management** | **List of stakeholders priortised by their importance to the project** |
| Identify project constraints | Interview senior management about constraints | Deadline, budget, predicted resource usage (i.e. employees, etc) of project |
| Identify risks | Interview senior management about long-term plans | A list of prioritised risks, based on likelihood as informed by management |
| Identify business problems caused without the presence of the API | Interview product owner about the benefit of the API to the business | A list of business priorities to be satisfied by the API |
| | Shadow a person whose work will be impacted by the API | A list of technical priorities to be satisfied by the API |
| **Identify process requirements** | **Interview senior management about process utilised** | **Considerations such as release schedule, possible requirement sources and team structure** |
| **Technical Requirement Collection** | | |
| **Determine technical constraints** | **Interview product owner & technical managers about interoperability standards** | **Whether or not the API is accessed internally/externally, the communication protocol utilised and the data-structures used.** |
| | **Interview product owner & technical managers about development standards** | **Implementation languages, style-guides that have to be followed and other patterns that have to be adhered to during the development process** |
| **Determine parallel deployment needs** | **Review API consumer personas, team structure & project constraints** | **Whether or not parallel deployment is necessary & its purpose** |
| **Functional Requirement Collection** | | |
| Determine the business asset to be exposed | Interview product owner & lead developer(s) | Broad, abstract list of data (in case of data being exposed) / functions (in case of logic being exposed) exposed by the API |
| Determine the operations to be conducted on the business asset | Interview product owner & lead developer(s) | List of user stories based on different functions to be conducted by the API in the form of "As an App SW developer who is/has ... I want to ... because of ... ". **This represents the API model.** |
| Determine the granularity of the API endpoints | Review API consumer personas & user stories | List of endpoints with their arguments & responses. **This represents the API specification.** |
| **Non-Functional Requirement Collection** | | |
| Determine appropriate levels of Performance | Workshop with all stakeholders present | A set of computational needs for the API (i.e. hardware needs, resource needs) |
| Determine appropriate levels of Uptime | Workshop with all stakeholders present | The hours of the day the API should be available, and during which days |
| Determine appropriate levels of Security | Workshop with all stakeholders present | Security protocols & methods to be utilised |
| **Determine appropriate levels of Usability** | **Workshop with all stakeholders present** | **The amount of control API consumers will have over the business asset with the exposed API** |
| Determine form of documentation | Workshop with all stakeholders present | Form of documentation |
| Determine maintenance & update needs | Workshop with all stakeholders present | Time-frame for maintenance, means of testing & risk of updates to the API |
| Determine appropriate levels of Scalability | Workshop with all stakeholders present | Projections for usage increase/decrease of the API |
| **Determine appropriate levels of testing** | **Workshop with all stakeholders present** | **Extent & means of testing in the form of coverage & patterns utilised** |

## VI. Iterations

Our regulative cycle consists of two iterations. The first iteration took place at SoftwareSkills AB, undergoing a participant observation developing a strategic API in order to gain insights into strategic API development and its relevance towards requirements elicitation. The second iteration involved the first implementation of the requirements elicitation method being integrated with API strategy considerations that arise from considering 'APIs as Digital Innovation Objects'.

### A. Iteration 1

For our first iteration, the goal was to primarily identify issues commonly faced in strategic API development and how these issues can be mitigated by effective requirements elicitation.

*1) Problem Investigation:* During this phase of the regulative cycle we conducted participant observation and in depth literature reviews. During the time at SoftwareSkills, we kept observation sheets for recording relevant information. We defined relevant as any activity or element that might influence the requirement elicitation process or the quality of the requirements produced. During this period, we gained several insights into strategic API development such as:

- It is important to identify API consumer skill levels
- It is important to identify bottlenecks in underlying systems/conduct source code analysis of the business artefact
- Open APIs require open data structures (i.e. JSON)
- Important quality attributes: security, performance, uptime, scalability
- Documentation is a high priority consideration for external APIs (including sample code!)
- Means of testing is important to determine
- Stakeholder identification & involvement is vital

This information collected towards the participant observation consisted of good API development practices and considerations to be made while developing. Due to the nature of the API we were producing, we were able to collect data from both the 'producer' side of the API and the 'consumer' side of the API. This was vital as it allowed us to gather information on API development practices which are lacking in existing literature. After the collection of our data from the participant observation, we analysed our review of standard requirements elicitation practices and determined which observations were relevant towards requirements elicitation for API development.

Parallel to the participant observation we conducted literature reviews regarding the following topics: APIs [1] [3] [4], requirements elicitation [7], digital objects [5] & Digital Innovation Objects [4].

*2) Design Candidate:* Based on the problem investigation conducted, we were able to identify several 'process areas' of requirements elicitation. While these areas do not necessarily have to be conducted in a sequential manner, they represent collection of similar types of information. Table 1 in section V.A presents the three different areas identified, with their primary objectives listed.

The initial/first process area we identified was the 'background information collection' area. The primary intent of this area was to identify stakeholders and their needs, project constraints, the business problem being addressed and any other long-term plans that might influence the project planning.

The second process area identified was the 'functional requirement collection' area. This primary goals of this area were to identify what the business asset being exposed by the API is, what functions/operations might be conducted on the business asset (i.e. in the case of an API serving images, the user might be able to ask for a specific size when requesting an image) and whether or not the API will be accessed externally (i.e. HTTP) or internally (i.e. application hooks).

The final process area identified was the 'non-functional requirement collection' area. The intent of this area was to collect information about quality attributes we found to be important specifically for APIs while conducting the problem investigation at SoftwareSkills AB.

*3) Validation:* To conduct validation of our design candidate, we approached our supervisor from University of Gothenburg who has expertise in requirements elicitation. While the design candidate was in a different format (i.e. not pictoral) the content was consistent throughout both representations. Through our validation we were able to refine the design candidate and begin implementation.

*4) Implementation:* In order to implement the design candidate, we assigned several tasks to each objective. The tasks were motivated with guidelines and suggestions as to how to conduct the task, types of questions to ask and what information might be relevant. Additionally preliminary interview subjects were suggested as a way to get started.

The tasks were typically interviews conducted with specific goals in mind. Additionally there was also 'group work' in the form of a workshop with all high priority stakeholders. Finally an ethnographic activity was added in the form of shadowing.

Zowghi and Coulin motivate that interviews are one of the most commonly used practises within requirements elictation [7]. They state that interviews are a efficient and enable large-scale collection of data at a fast pace. They identify three types of interviews: structured, unstructured and semi-structured. For our implementation, we chose to use semi-structured interviews by providing guidelines for how the interview is to be conducted. We believe this is the best choice as it allows the elicitor to adapt to real life situations and to gather information the interviewer did not anticipate. Zowghi and Coulin also state that interviews can be used for satisfying all of the requirement elicitors goals (as seen in table 2.1 [7] of their paper). We have chosen to use interviews for the following activities conducted by a requirement elicitor as defined by Zowghi and Coulin: understanding the domain, identifying sources of requirements, analysing stakeholders and eliciting requirements.

Zowghi and Coulin state that group work is useful as it enables collaboration between a wide variety of stakeholders [7]. We chose a workshop with all high-priority stakeholders present as the means for eliciting non-functional requirements

since there is a high risk of conflicts between the desires of the stakeholders in this regard. There was little purpose in having one task per objective as they are quite granular, therefore it was suited towards a large-scale collaborative activity. Additionally it is resource intensive and time consuming to host such large-scale collaborative events, therefore only one workshop as suggested. We have chosen to use the workshop for the activity of eliciting non-functional requirements.

Finally, Zowghi and Coulin write that ethnography is a useful technique for understanding contextual factors [7]. They motivate that it is particularly effective when a new system is being built to address existing problems. Therefore we chose shadowing as a method for understanding the domain.

Through our participant observation and additionally expert opinion from a requirements elicitation expert, we were able to determine that personas were a good way to categorize large groups of users [16]. Typically API consumers (particularly for open APIs) fall into groups, either due to design (i.e. pricing tiers) or due to other contextual factors (i.e. beginner and expert programmers using the API). Therefore personas are an appropriate way to categorize users in regards to their expertise and desired functionality. The knowledge in expertise can be used to drive factors concerning usability and granularity of the endpoints as described within the requirements elicitation method.

User stories were chosen as they are an effective and concise way to express functionality [17]. Written from the perspective of application developers consuming the API, will help inform concerns regarding the business asset, the features it exposes and how it is done. This artefact can be used to predict and aid in the determination of the end-points of the API as described in the requirements elicitation method.

*5) Evaluation:* The evaluation of our first iteration of the regulative cycle was conducted through the use of expert evaluation [14]. The experts were chosen based on their previous work within the field of strategic API design. We conducted three recorded interviews, having given them the design artefact and a set of interview questions beforehand.

### *B. Iteration 2*

*1) Problem Investigation:* The three interviews conducted for the evaluation of iteration one (as seen in section VII.A) also acted as our problem investigation for iteration two. Alongside evaluation questions, the interview instrument sent out to the experts also had questions regarding standard API development practices, techniques and the influence the 'APIs as Digital Innovation Objects' model has on requirements elicitation practices.

One of our findings from the problem investigation indicated that often sources of requirements are highly distributed throughout a company - in regards to the business asset (specifically the business model surrounding it) and additionally functional requirements. One way to tackle this as an elicitor is to simply ask interview subjects who they feel would be able to best meet the objectives of the different tasks, similar to 'snowball sampling' - an established statistical method [15]. Another finding was that API team size can largely affect the process used which can have implications on the elicitation process.

As mentioned in the evaluation of iteration one, parallel deployment of multiple APIs was seen as a common methodology employed to address issues related to versioning, tiered access, remote team access and many others. A finding from our problem investigation was that the main factor when considering the possibility of multiple APIs was the cost-benefit analysis of deploying it. This simply means to measure whether the cost of deploying, maintaining and hosting an additional API will be outweighed by the profit of having multiple APIs. This profit can be measured in several ways: decrease in development time, the revenue from the users of the API being greater than the cost of running it, the benefit to consumers of the API being able to maintain use of legacy versions and many others. We found this to be difficult to determine, as the cost of maintaining multiple systems (depending on the standards of testing & maintenance involved) can grow exponentially.

One surprising finding during the interviews was that the most common roadblock faced during API development is the release time for an API. Often, companies are unable to establish when exactly the API being developed is suitable for external/production use. The problem factor seems to be when companies are unable to determine when enough of their business asset(s) are being exposed to be useful alongside functionality concerns (i.e. do the endpoints behave correctly). The interviews suggested company culture to be the largest dictator of when releasing is appropriate - for example agile companies might be more suited to release the API in its smaller, earlier form while companies working with open-source development might choose to release the API early in order to get community feedback. Alternatively companies developing critical APIs (such as for use in medical systems) might choose to release the API in a much more complete form.

In regards to 'APIs as Digital Innovation Objects', we found the domain layer to be the most important towards requirements elicitation during the interviews. The implications of this were that the end users of the application software being implemented to consume the API become high priority stakeholders within the elicitation process. This was seen as difficult to achieve from the perspective of the API producers, as the end users are separated by the App SW layer of the 'APIs as Digital Innovation Objects' model. Therefore we have determined that it is vital to include users as early as possible within the elicitation process - and if not, at least API consumers who will build applications with the API. Additionally we found that the subsequent layers (App SW, API & Business Asset) follow the same order of relevance towards requirements elicitation.

One of the largest findings of the problem investigation was the realization that the information being collected with our requirements elicitation method lends itself directly towards creating the boundary objects specified by the 'APIs as Dig-

ital Innovation Objects' model. For our method, this means producing the API specification and the API model. One interesting finding was that our method was already generating the API model through objective two in the functional requirement collection process area and generating the API specification through objective three.

*2) Design Candidate:* In order to address the problems identified in the evaluation of iteration one, many changes were implemented on the design candidate. In the 'Background Information Collection' process area the stakeholder investigation objective was modified to encapsulate prioritization of stakeholders in regards to their involvement with the business and the cost associated with losing that stakeholder. Additionally, identification of process requirements and business model considerations were also added as additional objectives under 'Background Information Collection'.

An additional process area "Technical Requirement Collection" was added to cover technical needs such as implementation language, interoperability standards or other standards that might have to be adhered to. Determining usability needs was explicitly added as an objective of the 'Non-functional Requirement Collection' process area. Additionally testing concerns were explicitly stated as their own objective, while before they were encapsulated under maintenance/updating needs.

Furthermore a 'dependency visualization' depicting a rough chronology method was created. This visualization provides suggestions towards objectives that should typically be completed before conducting other objectives, as often the outputs of different objectives can be used for use in determining other objectives.

*3) Validation:* Through examining the activities and responsibilities of a requirements engineer defined by Zowghi and Coulin [7] we were able to validate our second design candidate.

*a) Understanding the application domain:* We satisfied coverage of understanding the application domain through analysing the business model, identification of long-term plans and risks and identification of business problems caused without the presence of the API. While there are a large spectrum of application domain concerns that differ from project to project which cannot be addressed by a generalized method (i.e. the problem domain the solution is being utilised within) the API specific and generalized requirements elicitation concerns were addressed.

*b) Identifying the source of requirements:* Identification of sources of requirements were conducted within the method through identification of stakeholders and the business problems caused without the API. This was encapsulated in ethnographic activities such as shadowing or source code analysis as well as interviews. All interviews had a suggested subject, however snowball sampling was recommended as a technique since typically requirement sources are not uniform and can be highly distributed within an organization.

*c) Analysing the stakeholders:* Analysis of the stakeholders was achieved through their identification and prioritization. The personas generated represent API specific stakeholders, while more typical stakeholder identification and analysis is conducted through the identification and prioritization of non-API-consuming stakeholders.

*d) Selecting tools, techniques and approaches:* For all tasks of all objectives within the requirements elicitation method activities to be conducted had guidelines as to what information to collect, suggestions for questions to ask regarding interviews alongside suggestions on subjects of said interviews. The guidelines and suggestions within the tasks often covered API-specific needs and information that need to be collected. The activities suggested by the requirement elicitation method are: interviews, ethnography and group-work.

*e) Eliciting requirements:* The majority of the objectives and tasks proposed in the method deal with eliciting requirements. The group workshop proposed for the purposes of non-functional requirement collection is one example. All of the outputs of the tasks within this process area can be translated into requirements for the system. Furthermore the functional requirement collection process area also involves directly eliciting requirements. These requirements are rather API-specific and involve considerations regarding the business asset, the functions conducted upon the business asset and the level of control over the business asset afforded to API consumers. While the guidelines and suggestions within the tasks of this process area were very API specific, the tasks conducted to achieve the outputs were drawn from current practices regarding requirements elicitation. The technical requirement collection process area also involved conducting elicitation of requirements regarding technical contextual factors.

In conclusion, the second design candidate for the requirements elicitation method managed to cover the activities specified by Zowghi and Coulin in their most generalized form. While there are many project specific considerations to be made which cannot be encapsulated within this generalized method, API specific considerations were also motivated and used to provide guidelines for the tasks to be conducted.

Furthermore the second design candidate had greater coverage than the first regarding requirements elicitation practices and API specific needs through the integration of the findings from the evaluation of iteration one alongside the problem investigation of iteration two.

*4) Implementation:* Several tasks were added and modified to the method in order to cover the changes within the design candidate.

Identification of the business model, identifying process requirements and determining technical constraints were all chosen to be satisfied through the conduction of interviews. The reasoning behind this was similar to the previous iterations: these pieces of information are highly problem domain, application domain or project specific. Therefore a high level of flexibility was seen as important in order to collect these pieces of information

Furthermore in order to address conflicting requirements from different sources we chose to conduct a prioritization

of stakeholders through a cost-benefit analysis of rejecting requests from a given stakeholder. This was chosen as it is a concise way to rank stakeholders in regards to their importance to the project. Addressing parallel deployment needs was chosen to be satisfied through reviewing outputs of several previous objectives. As there are several factors that go into determining whether parallel deployment is appropriate for a given project therefore the task conducted for this objective was a review of several artefacts generated by the previous objectives. These artefacts will directly inform the choice of parallel deployment, as elaborated within the requirements elicitation method.

Usability and testing were added as additional tasks of the group workshop. They were chosen as they are, similar to the other non-functional requirements, granular (i.e. target a specific area) in nature and present great opportunities for conflicts between the stakeholders. By consolidating them within the group workshop they can be addressed in a collaborative fashion with all high priority stakeholders.

Additionally several of the pre-existing tasks were modified to be more API specific or provide more guidelines regarding how to meet the objective specified. Determining performance within the 'Non-functional Requirement Collection' process area was augmented with explicit consideration of throughput. Additionally source code analysis or database snapshots as means to examine the business asset were also suggested.

In order to address feedback regarding the aims of the tasks being confusing we chose to define the outputs into specific forms or pieces of information in order to ensure the elicitor obtains the correct information

*5) Evaluation:* The evaluation of our second iteration of the regulative cycle was conducted through a survey (results can be found in Appendix C) evaluating: the quality of the design artefact, advantages over standard requirements elicitation practices, coverage of API specific concerns alongside other qualitative factors regarding the requirements elictation method. The survey was sent out to ten participants, five of which were professional developers working with APIs from Bisnode AB, iCore Solutions AB and VisitGroup AB and the other five of which were students with API hobbyist experience.

## VII. Results

### A. Evaluation of Iteration 1

The evaluation of the first iteration of the regulative cycle was conducted through the use of expert evaluations. Experts were sent the initial implementation of the requirements elicitation method and design candidate before the interview.

*Interviewee 1*

The findings of this interview indicated that requirements elicitation method lacked coverage of some aspects of requirements elicitation. The first point mentioned was the lack of validation and prioritization of requirements (i.e. ensuring that requirements from different stakeholders do not contradict each other).

Another concern that appeared was the lack of emphasis on testing within the elicitation method. This implied that maintenance needs within our method were not adequately covered. We also found that versioning with respect to the possibility of using several APIs in case of legacy support, or new release strategies was lacking within our method. This additionally led us to discover that the requirements elicitation method lacked consideration of several contextual factors that can affect the API and its evolution. Factors such as process considerations (i.e. does the business employ use of an agile methodology), technical considerations such as API design patterns which might have to be followed (i.e. REST-ful APIs, CRUD APIs), business model considerations such as pricing tiers and international/company standards which might have to be adhered to.

Additionally the elicitation method also failed to address usability as a quality attribute important to APIs. Usability in this context is how easily the developers consuming the API feel it can be used.

Scalability was also found to be more important to address for external APIs where usage might be inconsistent (i.e. external, open APIs). The consideration of throughput (and additional performance indicators) in regards to performance was also lacking.

*Interviewee 2*

During the second interview, we found that three of the points from the first interview were reiterated. Both interviewee one and interviewee two stated that testing, versioning and process related concerns were not covered sufficiently by the elicitation method. Interviewee two provided an additional usage scenarios for having multiple APIs outside of versioning: having different APIs for different groups of users or distributed teams requiring an API per team.

Additionally interviewee two suggested analyzing the source code/inner mechanisms underlying business asset being exposed directly as a means to identify potential functional requirements.

*Interviewee 3*

Interviewee 3 exposed several concerns within the presentation of the requirements elicitation method. The primary issue was that the abstract view of the method failed to provide indicators as to what exactly within the objectives of the process areas was relevant towards requirements elicitation for APIs. Secondly the language used to describe the method also seemed to cause confusion. Words such as 'specification' carry significant meaning in the body of work of requirements elicitation - therefore when being presented the API specification as an output of one of the tasks of the method the reader might get confused as to what exactly the output might be.

Another finding was that the non-functional requirements collection process area was lacking in regards to flexibility when determining relevant quality attributes. Our initial implementation provided several quality attributes we defined as relevant for consideration, however it did not suggest to consider project-specific quality attributes that may arise during the elicitation process. For example, most APIs would

not consider distributedness as a relevant quality attribute, however APIs exposing torrent downloading and uploading functionality would.

Additionally, as interviewee two mentioned, source code analysis was suggested as well, however this time as a means to identify business problems caused without the presence of the API. Additionally screen recording/sharing was suggested as an alternative to shadowing in case the 'shadowed party' is unavailable.

### B. Evaluation of Iteration 2

In order to conduct a final evaluation of the requirements elicitation method, a survey was sent out to practitioners in industry, alongside students with relevant experience which contained a presentation on the requirements elicitation method, and several questions (results can be found in Appendix C).

The results from the survey indicated that most of the participants found the requirements elicitation method useful to some degree, with over 80 percent of the participants giving it a rating greater than three from a scale of zero (low quality) to five (high quality) when asked about its quality.

When asked to compare the requirements elicitation method to standard requirements elicitation procedures, some participants mentioned that they were unaware of how standard requirements elicitation procedures were conducted and were therefore unable to provide valuable feedback in regards to deviations. However, other participants stated that the requirements elicitation method was useful in providing a structured approach, covering appropriate areas and promoting explicit consideration of the business asset.

Furthermore the survey indicated that the requirements elicitation method provides some form of adequate coverage of API needs and requirements, with over 80 percent of participants rating the requirements elicitation method greater than three from a scale of zero (no coverage) to five (full coverage) regarding API specific considerations.

When asked if participants would use the requirements elicitation method within their own projects, participants provided a wide range of answers. One replied that the requirements elicitation method required a execution within a 'live project' before it was suitable for implementation within a company. Another motivated that it seemed time consuming to conduct the requirements elicitation method in its whole. Additionally one participant motivated that the requirements elicitation method was unclear as to which use-cases the requirements elicitation method is appropriate for (i.e. for a new API strategy, or for every API developed) and finally another participant stated they would implement it if it provided a greater focus on the business use-cases of the API.

Regarding additional insights that the requirements elicitation method provided that elicitors would typically not consider, participants mentioned: the use of a dependency visualization between different objectives, the use of personas to model API consumers and the consideration of the business asset. Additionally another participant mentioned that the requirements method was useful in regards to ensuring that key factors are 'kept in mind' when eliciting requirements however this not necessarily an insight that was new.

Finally, when participants were asked if they had any thoughts on how the requirements elicitation method could be improved many participants stated changes that were already covered by the method. This indicates that some parts of the requirements elicitation method (particularly the summarized version presented to participants) are unclear, the most prominent one being the agile nature of the requirements elictation method presented. Two participants motivated that the method seemed to be 'waterfall' in nature and lacked agile principles. Secondly another participant mentioned that considerations about future versioning were lacking within the requirements elicitation method, however these considerations were present within the detailed implementation. Furthermore two participants stated that it was unclear for which size and type of API project the requirements elicitation method was to be used for. Finally, one participant mentioned that the requirements elicitation method required an even simpler summary for introduction.

Overall, the survey results indicated that the requirements elictation method was adequate in providing a good quality, structured, API-specific set of practices which provide an acceptable level of coverage in relevant areas of requirements elicitation. We were able to determine some ways in which the requirements elicitation method provided greater utility over standard requirements elicitation practices. Additionally we were able to identify several areas in which the requirements elicitation method was lacking, or unclear. While the results are useful for qualitative and quantitative analysis, the survey would have benefited from having more participants with relevant experience within API development and requirements engineering.

### C. Differences from standard requirements elicitation

In this section we have listed what our findings from our participant observations and expert interviews indicate were specific to strategic API development.

- Background Information Collection

  *a) Identify business model:* Our findings indicate that while considering the business model is common within requirements elicitation, strategic API design considerations specifically typical API pricing strategies such as tiered functionality access, pay-per-call or subscriptions can lead to the generation of architecturally significant requirements and enable consideration of parallel deployment needs.

  *b) Identify stakeholders:* In regards to identifying stakeholders, our findings indicated that the generation of personas for consolidating large groups of API consumers is useful technique to identify different levels of expertise within the consumers in regards to their familiarity not only with development but also the solution domain in which the API is being utilised within. These personas are used consistently in several other tasks within the

method to inform several decisions such as identifying functional requirements.

*c) Identify project constraints:* Standard requirements elicitation practices provide adequate coverage of this objective in regards to APIs.

*d) Identify risks:* Our findings indicate that in order to better identify risks, changes within the 'Digital Innovation Object' layers should be considered. While the layers are not directly mentioned within the requirements elicitation method, risks due to changes in the different layers (i.e. changes in the business asset or changes in the domain layer impacting API consumers) are stated for consideration.

*e) Identify the business problems caused without the presence of the API:* Standard ethnographic activities alongside interviews provide adequate coverage of this objective in regards to APIs.

*f) Identify process requirements:* Typical requirements elicitation practices provide adequate coverage of this objective in regards to APIs.

- Technical Requirement Collection

  *a) Determine technical constraints:* While technical constraints exist for all projects, there are several which have a higher priority in regards to APIs. Our findings indicate that interoperability standards regarding communication protocols and whether or not the API is accessed internally or externally are key pieces of information to collect as these are vital requirements for all APIs.

  *b) Determine parallel deployment needs:* Parallel deployment, while a possibility for several kinds of projects, is explicitly important to consider for strategic API design. Our findings from our interviews indicate that there are several factors which which can drive this decision. These API specific factors have been encapsulated within the method: versioning, pricing tiers/strategy, API consumer groups and distributed API teams.

- Functional Requirement Collection

  *a) Determine the business asset to be exposed:* As all APIs deal with exposing business assets, as expressed by Hammouda et al [4], determination of a business asset in the form of data or logic/functionality is a specific practice in requirements elicitation for APIs. Our findings indicate that as all functionality regarding the API is directly tied to the business asset, concretely determining what the business asset is a vital and critical step to undertake for requirements elicitation for strategic API design.

  *b) Determine the operations to be conducted on the business asset:* Our findings show that determining what operations the business asset will undergo is specific to requirements elicitation for strategic API development. As many APIs not only expose a business asset but also impose some form of functionality on them - particularly for data driven APIs (i.e. image resizing for an image requesting API) - it is important to determine what these operations are in order to reason about how the API will function. These conclusions also inform developers on what functionality needs to remain inaccessible through the API - something that typically is specific for strategic API development.

  *c) Determine the API end-points:* Alongside determining the business asset and what functionality the business asset is to expose, determining the end-points themselves through considerations of how much control needs to be afforded to the API consumers over the business asset (by examining API consumer personas and usability determinations) is also specific to requirements elicitation for strategic design of APIs. This is unique to APIs as often the interface for a system (i.e. exposed methods) would not typically be so explicitly determined by the elicitation process.

- Non-Functional Requirement Collection

  *a) Determine appropriate levels of performance:* While performance is a consideration for almost all requirements elicitation's, our findings indicate for APIs often the main factor which determines appropriate levels of performance and whether or not those are achievable are the underlying business asset itself. Consideration of the throughput of the business asset is vital as it allows the elicitor to address concerns regarding appropriate response times for the given business asset.

  *b) Determine appropriate levels of uptime:* Our findings indicated that the considerations regarding uptime were not very specific for strategic design for APIs and standard elicitation practices provided adequate coverage.

  *c) Determine appropriate levels of security:* Although security is often considered as an important quality attribute for most software projects, it is distinctly different for APIs. APIs can typically be divided into two categories: internal or external, and importance of security changes for each of those categories. Internal APIs (i.e. accessed locally on the same machine, or on the intranet, etc) do not require high levels of security, whereas external APIs open to the internet definitely require consideration of API access protocols such as secret keys or OAuth, etc.

  *d) Determine appropriate levels of usability:* While there are not many specific concerns regarding strategic API design in terms of usability, our findings indicate the use of the API consumer personas for the purposes of determining the 'ease of use' of the different end-points (i.e. the learning curve) to be specific towards requirements elicitation for strategic development APIs.

  *e) Determine form of documentation:* Our findings indicated that there were considerations regarding documentation were not very specific for strategic design for APIs and standard elicitation practices provided adequate coverage.

  *f) Determine maintenance and updating needs:* While the means of updating and maintenance are important for any software project, they are especially important considerations to make for strategic API development.

As this section involves making determinations regarding backwards-compatibility and versioning practices to be utilised (i.e. the previous three versions of the API need to be available) the findings here can further influence requirements regarding parallel deployment and how the different instances of the APIs will be maintained.

*g) Determine appropriate levels of scalability:* Our findings indicated that there were considerations regarding scalability were not very specific for strategic design for APIs and standard elicitation practices provided adequate coverage.

*h) Determine appropriate levels of testing:* Our findings indicated that there were considerations regarding testing were not very specific for strategic design for APIs and standard elicitation practices provided adequate coverage.

## VIII. Discussion

### A. Research Questions

*1) What is specific to requirements elicitation for strategic design of APIs?:* Our findings from the participant observation and subsequent interviews with experts indicate that there are several considerations that need to be taken into account when conducting requirements elicitation for strategic design of APIs. Overall we found that considering API specific & API strategy specific needs did not necessarily add *new* activities or responsibilities for a requirements engineer outside of the ones defined by Zowghi and Coulin [7], however in most tasks conducted by an elicitor API strategy specific considerations change the *way* in which these tasks are done and offer *guidelines* towards the information to be collected. The list provided in VII.C iterates through all of the objectives identified within the requirements elicitation method, and what is specific for the strategic design of APIs in each objective.

Consideration of APIs strategy concerns also enabled the requirements elicitation method to define and collect several architecturally significant requirements for APIs. Firstly, the potentially biggest architecturally significant requirement being whether or not the API is accessed internally or externally - both have major implications on the communication protocols utilised and the architecture of the overall system. Secondly parallel deployment requirements as these can also have a big impact on the architecture of the system and the way in which it is hosted. Finally, all of the considerations within the functional requirement collection process area of the requirements elicitation method lead to the definition of architecturally significant requirements as the business asset and the decisions made regarding the way in which the business asset is used by API consumers will be one of the major decisions to take into account when designing the architecture of the system.

These findings are corroborated by results from the survey conducted for the purposes of evaluating the final design artefact in the second iteration of our regulative cycle. Most participants indicated the requirements elicitation method provided good coverage of API specific concerns. While the survey was by no means conclusive with more empirical data required, this is a good initial indicator towards the validity of the API-specific needs identified by the requirements elicitation method.

*2) How can insights from considering APIs as Digital Innovation Objects improve requirements elicitation for APIs?:* Our findings indicate that there are several improvements in requirements elicitation for APIs when considering insights derived from considering 'APIs as Digital Innovation Objects'. We have considered improvement in this context to aid in achieving any of the activities or responsibilities for a requirements engineer and additionally leading towards correct and valid requirements.

The primary benefit towards requirements elicitation by considering 'APIs as Digital Innovation Objects' are the impacts the four emergent layers (business asset, API, App SW and domain) have on the elicitation process. Considering the API as a layer between the business asset and application software developers allows for a clearer definition of functional requirements and the means to represent them. By viewing the data or functionality exposed by the API as a "business asset", the requirements engineer can analyse the business model and potential changes surrounding the business asset to better deliver a correct set of requirements which can support these needs.

Outputting the boundary objects adjacent to the API layer within the APIs as Digital Innovation Objects model within the requirements elicitation method enabled a clearer definition of functional requirements. By taking this 'double-sided' approach to API development a requirements engineer can more accurately map the functionality that the organization wishes to expose from the business asset towards the way in which API consumers will digest these features.

Aiming to create the API model, defined by Hammouda et al [4] as the boundary object between the business asset layer and the API layer, enables a requirements engineer to better determine exactly what aspects of the business asset are to be exposed. While the requirements elicitation method proposed defines the API model to be in the form of user stories from the an application software developer perspective, this is not an established convention of representing an API model by any means but more-so a concise way to represent functionality through the same lens as the API model. The primary reason for this was the lack of formal definition of API model within the 'APIs as Digital Innovation Object' model. Although application software developers exist on the layer above APIs (App SW layer), it was chosen to use their perspective when writing the user stories as they represent the end-users of the API.

Similar to the API model, aiming to create the API specification, defined by Hammouda et al [4] as the boundary object between the App SW layer and the API layer enables a requirements engineer to explicitly state the contract regarding the API towards API consuming stakeholders (i.e. App SW developers). The contract, in this context, specifies the *way* in which the relevant aspects of the business asset are utilised (i.e. the API model). Similar to the previous boundary object an API specification lacks a formal definition, however we

represented it through a form of documentation containing the API end-points, the arguments they take and what the responses are. This is useful for a requirements engineer as this form of documentation is necessary to enable API consumers to use the API effectively.

Alongside defining key formats for collecting functional requirements, considering the layered view of APIs from the 'APIs as Digital Innovation Objects' model put a greater emphasis on considering risks regarding changes within the layers. As Hammouda et al motivate, changes within boundary objects lead to changes within adjacent layers [4]. By considering and continually refining these boundary objects as part of the elicitation process, a requirements elicitor can be aware of potential changes that might impact the API.

*3) To what extent is it feasible to consider API strategies for driving requirements elicitation?:* By considering what is specific to requirements elicitation for strategic design of APIs as discussed in VII.A.1, we can identify that there are a many areas of requirements elicitation that can be affected by API strategy considerations. The main impact on the elicitation process of these considerations is the way in which tasks are carried out by a requirements elicitor as opposed to the tasks themselves. Our findings indicate that API strategy considerations can promote respect for business, technical and process related factors that are important for the project. By considering these factors, a requirements engineer may be able to ask more direct and appropriate questions in order to drive the requirements elicitation. This in turn can lead towards the definition of correct requirements in the sense that they align with the goals of the organization, the constraints placed upon the project and the technical domain within which the project takes place.

As Chen et al [8] state, identification of architecturally significant requirements typically necessitate some level of solution space knowledge regarding the product. Considering API strategies can expedite the acquisition of solution space knowledge by providing a requirements engineer with key considerations to be made regarding business, technical and process related factors. Consideration of these factors will typically lead towards the definition of architecturally significant requirements.

Furthermore, consideration of API strategies did not add significant new areas for a requirements engineer to analyse outside of those defined by standard practices. Consideration of functional requirements, non-functional requirements and quality attributes, identifying background information and determining technical constraints are tasks that are typically carried out for most software projects. However, API strategies can enable a requirements engineer to directly acquire necessary information as opposed to conducting investigations to determine what the important factors are for a project and then acquiring this information.

Throughout this study we have utilised the 'APIs as Digital Innovation Object' model to inform concerns regarding API strategy. By considering the layers that emerge from this view (business asset, API, etc) we were able to conceptualise API strategies through consideration of the concerns that arise within these layers. Our findings indicate these concerns directly informed several aspects of the requirements elicitation as discussed in VIII.A.1 and VIII.A.2. Therefore we can determine that consideration of API strategies and even models which are used to reason about API strategies can have some impact on requirements elicitation.

The survey conducted for the evaluation of the requirements elicitation method produced in this study indicated that most participants found the model useful in some regard. Participants found API specific considerations to be covered to an adequate extent with more than 80 percent of them giving it a rating greater than three out of five. Additionally participants wrote that the requirements elicitation method provided greater structure, coverage of API specific needs and useful artefacts to use during the elicitation. This implies that the consideration of API strategies for driving requirements elicitation is indeed feasible as it provides practitioners with some level of utility. While our initial findings indicate positive results, there is a need for testing the requirements elicitation method within an industrial environment in order to better understand the extent to which these strategic insights improve requirements elicitation when compared to standard practices.

### *B. Threats to Validity*

We utilise the definition of validity threats provided by Runeson and Höst [18].

*1) Construct Validity:* As the requirements elicitation method contains detailed explanations of all the tasks provided, this makes it rather time-consuming to read. It is possible that many participants in the survey (even in the expert interviews) did not take the time to go through the detailed implementation of the requirements elicitation method before conducting their evaluation. This is reflected in some of the survey answers, where participants asked for changes or made suggestions that were already considered at some stage within the requirements elicitation method.

Additionally survey participants were chosen by the authors as they were required to have at least some knowledge of one of the following: APIs, requirements engineering or development in order to obtain meaningful results regarding the evaluation for the requirements elicitation method. As they were associated with the authors, they might have been biased in their feedback based on what is acceptable or expected. In order to alleviate this we ensured all the data collected was anonymous and stated this to the survey participants.

*2) Internal Validity:* Due to time constraints, we were unable to conduct the requirements elicitation method produced within a company. Although it has undergone evaluation by several experts and practitioners within industry which indicate the requirements elicitation method provides some level of utility, it still remains to be fully concluded that the method is appropriate for use within the real world and that it achieves its purpose.

Additionally survey participants were chosen due to their experience within the field of API development, requirements

engineering or strategic API design. There was a large skill gap between the students approached for participation and the industrial practitioners. This might influence the survey results especially in regards to new insights the requirements elicitation method provides as these individuals have likely already gathered this knowledge, or lack basic knowledge in the field of requirements elicitation. In order to mitigate this we collected demographic data regarding experience with APIs as part of the survey.

*3) External Validity:* The participant observation conducted was only done within a single company. While the artefact produced in this study is designed to be applied in any strategic API development project, conducting participant observations or case studies in other API development projects regarding requirements elicitation for strategic API development will greatly help in further generalizing the results and the requirements elicitation methodology produced.

Additionally all experts identified within the fields of requirements engineering, 'APIs as Digital Innovation Objects' and API strategy were from the same institution. There might be other schools of thought regarding conceptualization of API strategies using models other than 'APIs as Digital Innovation Objects' that should be considered to provide a more generalized method and results.

*4) Reliability:* The participant observation was conducted within a company in which one of the researchers is employed. This provides an advantage towards the researchers in terms of familiarity with the company. While not a big factor, it enabled direct and more efficient communication regarding the initial problem investigation. However, other researchers aiming to replicate the study can overcome this barrier by investing time into becoming familiar with their case company.

As the participant observation involved working on a API project, researchers with greater industrial experience with development might be able to extract more insightful and accurate information as they will have knowledge regarding implementation and design issues to do with the API and be able to focus more on the sections relevant towards requirements elicitation.

## IX. Conclusion

This study examined the implications of considering API strategies on requirements elicitation. The methodology used was design science research with the design artefact being a requirements elicitation method informed by stategic API design considerations. During the study, a participant observation was conducted on a live API project being executed in industry. The participant observation helped provide insights regarding API development and what areas of strategic API design are relevant towards requirements elicitation. Additionally interviews with experts were conducted in order to further understand the problem and evaluate progress. In order to determine how valid the requirements elicitation method produced was, a survey was sent out to practitioners and students with relevant experience. While the results of the study indicate that the elicitation method produced is useful to some extent and provides insights that practitioners would not typically consider, further measures such as adding more participants to the survey or testing it within a real project can be taken to concretely confirm how the requirements elicitation method provides greater utility than standard practices. Our results indicate that API strategy considerations can aid in requirements elicitation and consideration of 'APIs as Digital Innovation Objects' provide a useful framework for reasoning about API strategy.

### A. Future Work

There is definitely a need for further research within this area. Empirical studies gathering further quantitative data to prove the applicability of API strategies in requirements elicitation are required to draw concrete conclusions. Additionally API strategy considerations should also be investigated for their relevance towards other areas of the development lifecycle, outside of requirements engineering. Other models for reasoning about API strategy outside of 'APIs as Digital Innovation Objects' should also be investigated as they might prove to be more relevant towards requirements elicitation. Regarding the design artefact produced within this study, there is a need for executing and evaluating it within a industrial setting in order to determine its practicality and confirm the findings presented in this study.

## Acknowledgment

We would like to thank Eric Knauss for his valuable feedback, advice and insights he has given us. We would also like to thank Imed Hammouda, Juho Lindman and Jennifer Horkoff for letting us pick their brains. Additionally we would like to thank SoftwareSkills AB, particularly Henrik Enström for accommodating this study.

## X. Appendix

### A. Requirements Elicitation Method for strategic API design

**Background Information Collection**

1) Identify business model
   a) *Interview senior management & product owner*
      The intent of this task is to identify the business model surrounding the business asset(s) exposed by the API. It is important to discuss with senior management and the product owner the ways in which the business asset will generate value. This information will later inform collection of all other requirements, as they must all support these goals.
      Typical business models surrounding APIs include pricing tiers for different levels of access (i.e. rate limitations) or different levels of functionality (i.e. blocks of functionality require paid access). Factors such as these can greatly influence the definition of requirements. One example of where the business model can impact requirements is the pricing strategy - if the price is calculated on a per-call basis as opposed to a subscription fee.
      It is vital to obtain a wholesome view of the business model as this is one of the primary ways to ensure consistency throughout the requirements. All requirements collected should support the business model identified. Any requirements which oppose the business model should be investigated as they contradict the companies needs in regards to the API. The output of this task is a description of the business model surrounding the business asset(s) to be exposed.
2) Identify stakeholders
   a) *Create Personas for API consumers with product owner*
      The intent of this task is to identify different groups of API consumers that will use the API, for the purposes of later dissecting how they might wish to use the API. The personas should all encapsulate what the aim of their usage of the API is, expertise and knowledge of the consumer in both the development domain and the problem domain (i.e. optics, if the API is for a telescope), what they do and do not find important while using an API and typical pain points for their usage of the API. Typical groups of users for APIs that should have personas are: enterprise-level consumers (i.e. businesses) and hobbyist-level consumers (i.e. hobbyists or small teams). If the API is only to be accessed internally (i.e. within the business), the latter group should not be considered. Otherwise, **external** enterprise-level API consumers should additionally be considered. The output of this task will be a list of personas encapsulating the typical API consumers to be expected.
   b) *Identify other relevant stakeholders with product owner*
      The intent of this task is to determine stakeholders who are relevant outside of consuming the API. Relevance is determined by the amount of control they have over the project and by the amount of impact they can exert on the development effort. For each of these stakeholders, their desires, experience within the company and relevance to requirements elicitation and the API design should be identified. For high priority stakeholders, it is advised to interview them directly for collection of this information. Typical non-consuming stakeholders include: developers who will produce the API, parties funding the development, developers working on the underlying logic being exposed by the API, regulatory bodies such as governments and any relevant activist groups. The output of this task will be a list of names with important stakeholder information identified.
   c) *Prioritise stakeholders with product owner and senior management*
      The intent of this task is to prioritise identified stakeholders from the previous steps for the purposes of managing conflicting requirements. The primary method of prioritization of stakeholders should be based on a simple cost-benefit analysis of the cost to the business for rejecting possible requests in case of contradiction with other stakeholders. This is a vital step to undertake, and while it may sound simple, it is difficult to quantify the cost & benefit for different stakeholders as they might impact the project in unforeseen ways. The output of this task is a list of prioritised stakeholders.
3) Identify project constraints
   a) *Interview senior management about constraints*
      The purpose of this task is to determine what business constraints the project will operate under. The primary properties to identify are the available time for development, budget and resources at hand. This can be done through interviewing senior management (or typically any party managing the project). This information will determine whether or not the project will be within a reasonable scope. The output of this task is the deadline, budget and predicted resource usage of the project.
4) Identify risks
   a) *Interview senior management about long-term plans*
      The purpose of this task is to identify long-term plans that might have an impact on the project. It is important to identify, from anyone with the relevant information and right position in the hierarchy, what the future plans for the API are, what the plans for the business asset being exposed by the API are, what the future plans for product(s) using the API are, possibility of the company pivoting the API (i.e. changing the core functionality), possibility of disruptive technology release (i.e. a new compression algorithm which requires a refactor to integrate) or employees leaving with important knowledge. The output of this task should

be a prioritised list of risks, based on their likelihood of occurrence as informed by the management.

5) Identify the business problems caused without the presence of the API
   a) *Interview product owner*
      The purpose of this task is to determine what the benefit to the business is through the production of this API. This will drive all functionality created and is an important thing to keep in mind throughout the requirements elicitation process as contradictions might appear further into the process. In order to achieve this, the product owner should be directly interviewed. The output of this task is a list of business priorities to be satisfied by the API.
   b) *Shadow a person who will be impacted by the API*
      In order to fully understand what business areas the API will be impacting, it is also recommended to shadow a person whose job will be influenced by the release of the API, ideally someone who is the most impacted. During this shadowing, notes about what task(s) the API will influence should be recorded, what the current biggest roadblocks of those task(s) are, any structural changes that might have to be made during integration with the API and quirky behaviour that might have to be replicated. This is also a good chance to gain domain knowledge on the consumer side of the API. Alternatively if such a person is unavailable or too far away for direct shadowing, the elicitor may instead choose to view screen recordings / conduct screen sharing or review source code if the API impacts existing systems. The output of this task is a list of technical priorities to be satisfied with the API.
6) Identify process requirements
   a) *Interview senior management about process utilised*
      The purpose of this task is to identify requirements imposed upon the project due to the process used by the company. For instance, agile companies might have a weekly release cycle which would have to be followed. The size of the team working on the API alongside can be used for determining quality attributes such as the modularization. The number of teams working on the API can aid in determining parallel deployment factors such as providing the different teams with their own instance of the API (however additional factors have to be considered as seen in objective 2 of Technical Information Collection). Additionally the organizational structure of the project can direct the elicitor in regards to finding different sources of requirements. The outputs are process related considerations such as release schedules, possible requirement sources and the team structure.

**Technical Requirement Collection**

1) Determine technical constraints
   a) *Interview product owner & technical managers about interoperability standards*
      The purpose of this task is to identify international or company standards that dictate interoperability, alongside technical constraints that might impose some form of interoperability.
      It is vital to determine interoperability standards to be employed as typically these will dictate data formats and communication protocols to be utilised. For example, web-based APIs typically rely on the HTTP protocol for communication. Often, they utilise JSON or XML data formats for data exchange. Alternatively framework APIs might depend on application hooks and define their own custom data formats. It is important as the elicitor to determine what protocol and data formats are appropriate for the given project. Often, companies might have their own custom protocols and formats which might be mandatory to implement.
      By interviewing employees involved in the technical aspects of the API, the output of this task should be whether or not the API is accessed internally or externally, the communication protocol utilised by the system and the data structures sent back and forth.
   a) *Interview product owner & technical managers about development standards*
      The purpose of this task is to identify international or company standards that dictate development practices & process alongside technical constraints that might impose some limitations.
      Development standards are important to consider as they might drive many of the architectural choices made during development. For example, businesses might have standards dictating how much responsibility specific modules might be allowed to have. Alternatively companies might have in place strict style guides for programming which have to be adhered to. Additionally implementation language should be considered as the company might impose limitations in that regard. It is important to identify these standards early on as refactoring to meet them might be costly.
      By interviewing employees involved in the technical aspects of the API, the output of this task should be the implementation language(s), style-guides that have to be followed and other patterns that have to be adhered to within the development process.
2) Determine parallel deployment needs
   a) Review API consumer personas, API team structures & project constraints
      This is arguably one of the biggest factors an elicitor has to decide for an API. Based on several factors, the need for parallel deployment of multiple APIs might be necessary. Typically for small APIs with small teams, this is not a concern. However, for larger teams developing large-scale APIs exposing popular business assets (i.e. Google Maps API) it is important to consider the possibility of having several APIs.

Factors such as the different types of API consumers, the business model, development team structure and project constraints such as resources available all impact this decision.

In order to facilitate different groups of users and their needs, parallel deployment might be considered. A business might choose to have different APIs for different experience levels of consumers (i.e. a simple API for novice users and a more complex, flexible API for expert users). The business model is also highly relevant as businesses might choose to deploy different APIs for different price tiers. In case of several teams working on different parts of an API, the business might choose to provide them with individual instances of the API to develop. Additionally versioning is a big factor: in the event of an API update that changes the structure significantly, should a new instance of the API be deployed to enable backwards-compatibility for API consumers?

In the end, all of these factors come down to one key metric: the cost-benefit analysis. Is the cost of running a separate API greater than the value gained from doing so? This is a difficult question to answer, as often these costs and the value gained are very hard to determine. For example, having separate APIs for different pricing tiers might cost more in the short term, however ensuring the stability of the API for paying customers might provide more value than enabling greater access to free users. The elicitor should consult senior management on all these factors as they are highly relevant in this decision.

The output of this task is the decision of whether or not parallel deployment is appropriate for the project, and if it is, what purposes the parallel deployment will be used for (i.e. ensuring legacy user can operate the API, etc). These artefacts will enable discussions with senior management about the need for multiple API instances.

**Functional Requirement Collection**

1) Determine the business asset to be exposed
   a) *Interview product owner and lead developer*
   The purpose of this task is to determine specifically what business asset is being exposed. The business asset will come in typically one of two forms: data (i.e. images) or functionality/logic (i.e. natural language processing algorithms) - it is important to know precisely which one as it might influence the architecture of the API significantly. After obtaining this knowledge, it is also important to know how the underlying business asset is implemented. If possible, an instance of the business artefact (i.e. source code, or database snapshot) should be reviewed directly. This will greatly help in understanding the business asset and what functionality points are to be exposed. If it is data it might be enough to know what data storage is being utilised (i.e. NoSQL database, Redis, etc). The output of this task is a description/model of the business asset giving an overview of how it works and what it does.
2) Determine the operations to be conducted on the business asset
   a) *Interview product owner and lead developer*
   The purpose of this task is to determine what operations the business asset might undergo when being exposed by the API. In order to obtain a list of such operations, the product owner and lead developer should be questioned about what aspects of the business asset need to be exposed. For example, an image requesting API might allow for the consumer to request specific sizes, in which case the API needs to be able to re size the image being requested. It is important to determine what scope of functionality the API will provide over the underlying business asset exposed.
   The information should be recorded in the form of user stories. The output of this step is a list of user stories based on the different functions of the API in the form of "As a ... who is/has ... I want to ... because of ...".
   The user stories generated in this step dictate the **API model** of the system. The API model informs the API developers of the relevant aspects of the business asset to be exposed. This is key in ensuring that potential use cases from the domain layer are satisfied by the API and business asset exposed.
3) Determine the API end-points
   a) *Review the API-consumer personas and user stories*
   The intent of this task is to determine a list of endpoints for the system. For this purpose, the usability evaluation in Non-functional Requirement Collection task four must be completed. Based on the usability evaluation & the user stories, a list of end-points informed in: their *granularity* (i.e. amount of control) by the usability evaluation and their *functionality* by the user stories may be generated. In order to manage conflicts in terms of granularity (i.e. different API consumers want different levels of control) the prioritized list of stakeholders should be leveraged. Additionally in case parallel deployment is being considered, different APIs could cater to different granularity requirements. The output of this task is a list of endpoints, with their arguments and their responses per API.
   The list of API endpoints, with their arguments and output dictate the **API specification** of the system. This specification, alongside documentation can be provided to API consumers as their source for information about usage of the API.

**Non-functional Requirement Collection**

The following tasks are to be conducted during a workshop with all stakeholders present. It is important to utilise the stakeholder prioritization generated in Background Information Collection task 2.C in order to manage conflicting requirements. Additionally the quality attributes provided for

consideration are based off of general API practices - it is important to additionally consider project specific quality attributes that may arise as a result of this elicitation.

1) *Determine appropriate levels of performance*
   In order to determine appropriate levels of performance for the API, several factors have to be considered. The foremost is the the predicted request rate of the API - how many requests a day does the business anticipate on the API? Following this, the appropriate estimated response time for a request should be determined. For simple operations, this can typically be below 500ms. For more complex functionality, it might take up to hours.
   After the predicted request rate and an appropriate response time is defined, the computational power/time required to answer a request should be determined. If the business asset being exposed is logic/functionality, how much computational power/time does a typical execution of this functionality use? Additionally how much is the throughput of the system? Alternatively, if the business asset being exposed is just data, how much computational power/time does extracting the data from the storage require?
   Based on these two sets of data, the computational power needs of the API can be determined. If there is a small response time required and large computational power required to formulate a response to a request, the API should have access to large amounts of memory and CPU. The output of this task should be a set of computational resource needs by the API.
2) *Determine appropriate levels of uptime*
   There are multiple factors that need to be considered when reasoning about the uptime of the API, the largest of which is predicted peak hours for the API. If the API is an internal API only going to be used during working hours, it does not need to be available outside of those. However, if the API is critical to some businesses then it might need 100 percent uptime. The output of this task should be the hours of the day API should be available and during which days of the year.
3) *Determine appropriate levels of security*
   In order to reason about security, it is important to consider the following factors: what security measures the business already uses (they might be enough), and whether or not the API is internal/external. If the API is only going to be used within a local network (i.e. intranet), security might not be such a great concern. Otherwise, if the API is going to be exposed to the internet, there must be a minimum standard of security enforced.
   In the event that the API is open to the Internet, but has tiered access (i.e. users with different levels of authentication can access different endpoints) it is important to determine an appropriate authentication protocol, such as secret keys or OAuth, etc. The output of this task will be the appropriate security in the form of protocols/methods to be utilised.
4) *Determine appropriate levels of Usability*
   In regards to usability, the primary concern for the elicitor is the amount of control afforded to API consumers over the business asset and secondly how fast the API consumers are expected to learn to use the API. In order to achieve this, the API consumer personas and user stories should already be generated.
   By going through each user story generated covering the operations to be conducted on the API, the skill level and desires of each user in each user story should be identified through the personas. Novice personas will typically prefer simpler API designs, whereas expert users will typically prefer having more control. This can be referred to as the 'chattiness vs chunkiness' of the API - a chatty API will typically have more end-points allowing finer control over the business asset at the expense of increased complexity. Alternatively a chunky API will provide a simpler and fewer endpoints for controlling the business asset.
   For example, given an API for controlling a surveillance camera, a novice user might want to change the camera mode to night mode by simply making a request asking for night mode - however an advanced user might want to change the camera mode through modifying specific camera settings (i.e. red, green, blue levels being recorded).
   The output of this task will be the amount of control API consumers will have over the business asset with the exposed API. Specific end-points should not be determined, however the general level of control suitable for different API consumers for the API should be established.
5) *Determine form of documentation*
   As an API is a product for use by developers, it is important that documentation takes an appropriate format and is provided to the right people. If the API is an external, open API it is likely that the most appropriate form of documentation is web-based. If it is internal, it is advised to use existing standards within the company for documentation.
6) *Determine maintenance and updating needs*
   In order to investigate maintenance needs the primary property to identify is the expectation of how bugs will be addressed (i.e. contractors/consultants will be hired to fix them, the original development team will fix them, etc.). To determine how long maintenance needs to be available, it is important to ask how long the API will be in use, how often the underlying data structures will change and in case of an open API and does it have to keep up with the latest technology being released (i.e. image formats in the case of an image API).
   As the API will act as a link within a chain of software developed on top of it, it is important to address how and when the API will be updated early on in order to inform application software developers towards the strategy. This will in turn impact the development process they utilise. For critical APIs it might be important to maintain

backwards-compatibility in order to ensure new updates do not break implementations that are no longer being updated. For APIs exposing state-of-the-art technology requiring frequent updates, backwards-compatiblity might not be a concern. Additionally the point at which to update the API is also vital (and difficult) to determine - again, the API consumer personas and the list of prioritized stakeholders should be consulted (i.e. would making a big changes to features X, Y & Z impact high-priority stakeholders negatively?).

The output of this step should be a time frame for maintenance and the parties responsible for it alongside a strategy for future updates which can be communicated to API consumers.

7) *Determine appropriate levels of scalability*
   The need for scalability is primarily determined by whether or not the API is internal or external. In the event of an external API, forecasts regarding customers/users for the service should be asked for as they are likely the most accurate predictors for API usage. Additionally the likelihood of the API popularity exploding, resulting in a exponential increase in users should be noted. The output of this task is projections for usage increase/decrease of the API.
8) *Determine appropriate levels of testing*
   As with all software artefacts, it is important to consider what level of testing will be implemented. Even software project which have no specification for tests, are still tested manually by developers. Therefore, it is important to determine the extent to which the API should undergo testing. There are many factors that can motivate this: criticality of the system (i.e. medical systems), project constraints (i.e. budget available for testing) or whether the API is external or internal. There are several means of testing APIs, typical examples are unit test for end-points, integration testing for components that make up the API and regression testing to ensure that the system behaves correctly after changes. The output of this step should be the extent and means of testing in the form of coverage and patterns utilised.

*B. Interview Guide*

### Requirements Elicitation Method Evaluation

1. What is missing from our requirements generation model? In regards to:
    - Coverage of potential requirements
    - Coverage of standard R.E needs
    - Coverage of API specific concerns
    - Useful artefacts (such as personas, user stories, etc)
    - Other..
2. What areas have we covered appropriately?
3. What can be expanded on/improved?
4. What do you think would be the greatest obstacle implementing this model?
5. Any other comments you would like to make?

### General API/R.E knowledge questions

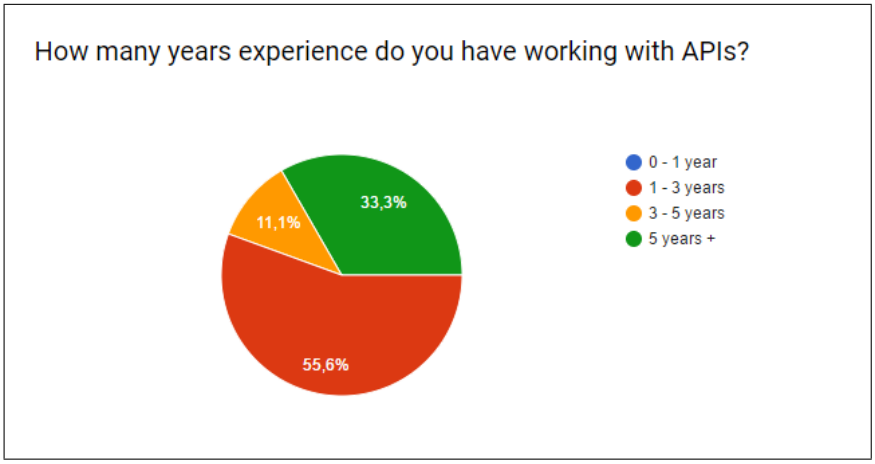
1. How can we structure an API so that it it can adapt to future changes?
2. What are the most 'generalized' considerations that need to be (i.e. for ALL APIs regardless of domain) made?
3. What different 'categories/genres' of API's, if any, are prevalent in industry? If so, are there any established considerations to make for these classifications?
4. Internal vs External API's: what are the different considerations to be made?
5. What are the high priority stakeholders to consider when it comes to API development?
6. Which 'R.E. activities' do you feel are most relevant to API development?
    - Prototyping, Interviews, Domain analysis, Etc .
7. Common API R.E/development roadblocks? (i.e. what do people often do wrong)
    - Some development issues can be traced back to incorrect R.E. What are common development issues you believe happen in the API development due to malformed requirements?

### API as DIO knowledge questions

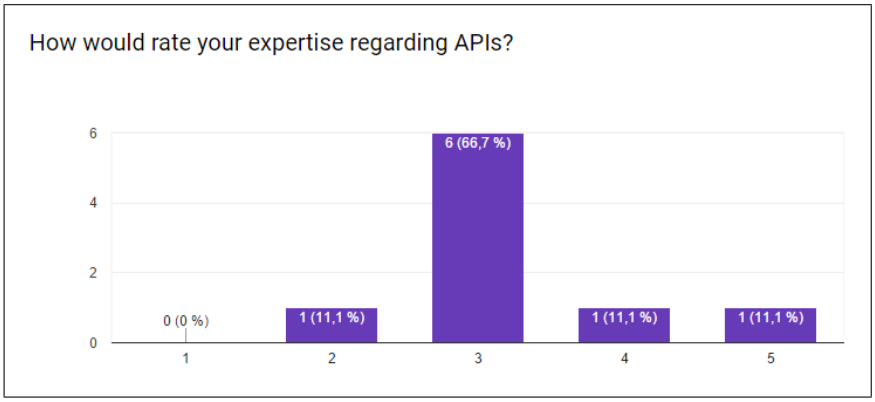
1. Rank the APIs as D.I.O layers in order of their relevance to requirements elicitation
    - Domain
    - App SW
    - API Model
    - Business
2. Rank the boundary objects of the layers in order of their relevance to requirements elicitation:
    - Use cases
    - API specification
    - API Model
3. What digital object attributes are most relevant to APIs? (most, to least)
    - Is the ranking the same for every API project (for example external and internal APIs)

*C. Survey Results*

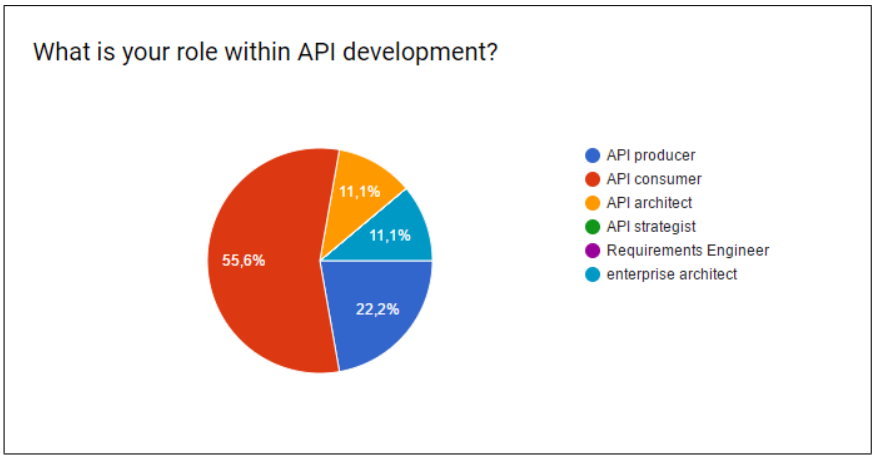
*Question 1*




*Question 2*

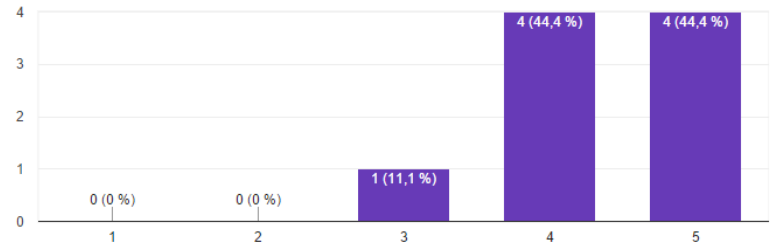



*Question 3*



*Question 4*

Rate the quality of the proposed requirements elicitation method



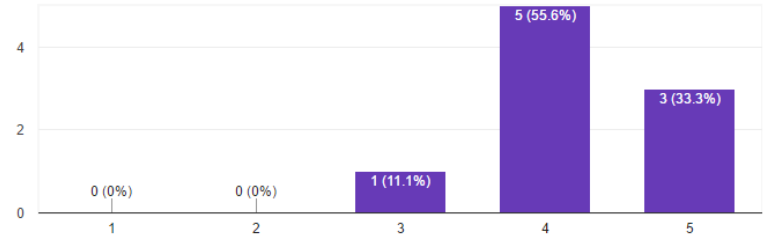

*Question 5*

Would you find this more useful than a standard requirements elicitation procedure? (Please motivate)

| |
|---|
| n/a |
| I am not sure, it´s a bit overwhelming in todays fast running world.But it covers a lot of great areas. |
| In my experience there is not much more beyond ad hoc meetings with stakeholders surrounding API requirements. This provides more structure |
| I'd use the most useful pieces from either of them for a particular situation |
| Due to the lack of knowledge concerning the "standard" requirements eliciation procedure, the question can not be fully motivated. |
| I don't know since I'm not familiar with a standard requirements elicitation. |
| This elicitation process allows a clear identification of the business asset, its consumers and how the asset should be exposed to them. |
| Yes, as it covers most areas of concern without diverging from an agile way of working. |
| About the same |

*Question 6*

To what extent are API specific considerations covered by the requirements elicitation method?




*Question 7*

Would you implement this requirement elicitation method in your company? (Please motivate)

| |
|---|
| Before it can be implemented it has to be tested and proven in a live project. |
| Maybe a part of it, but it seams to be a little time consuming. The question is if it is for totally new API strategy within a company or is it for every API a company develops? |
| Perhaps a version with more focus on determining use cases and business needs. A big roadblock in implementing APIs is providing a business case internally, they are typically a "cost" rather than a net source of revenue. It would be useful to include more on how an API could maximise business value. |
| I'm not in the position to do so. |
| Yes, in the "Non-functional requirement collection" - such as performance, usability, testing, scaling, security and usability - I get a feeling that everything is covered in order to reach the end-goal of perfect API. |
| Sure, why not. I'm not familiar with any other method and this one looks solid so I would give it a try. |
| API consideration |
| Absolutely. Nowadays there are no standard elicitation methods in use, which makes the quality of APIs vary to a great degree. |
| I would use this method or a similar method to gain an overview of the requirements |

*Question 8*

Did the requirements elicitation method provide you with insights that you would not typically consider? (Please motivate)

| |
|---|
| Using a flowchart to support the elicitation process. |
| No, I was aware of most of it. |
| The idea of using personas is an interesting and possibly valuable one. When not dogfooding the feedback loop is limited in API delivery so using personas might be a good analogue for that |
| It's a very good guide for all things to keep in mind and consider when examining requirements. |
| Yes, such as determining what business asset needs to be exposed. |
| I would say scalability. Since I've only been a student, the projects I've worked one have never required me to have scalability in mind. |
| see above |
| Yes, as I would typically rely on my opinion as a developer for selecting the API functionality. It is good to expand the scope and cover all possible stakeholders. |
| not really, its all common sense (to an experienced developer) |

*Question 9*

Do you have any suggestions on how to improve the requirements elicitation method?

| |
|---|
| Maybe discuss future versioning of the API's. In bigger projects "all stakeholders" can be quite a few people, not sure if all needed in all non-functional objectives |
| I belive that it should be more agile. Maybe more clearly how to be used in companies depending on size of solution (multiple API och one new?) |
| Have an even simpler summary to open that lays out the stages with some idea of the flow as well as the focus on business needs as above |
| It strikes me as being rather "waterfally". I believe a more iterative method could be useful. It is a little like a checklist. The method describes how (and with whom) information is gathered/elicited for each separate objective, but there's no process description how to coordinate actions, in which order to do it etc. Is the method appropriate for all types and sizes of projects, or should it be adjusted for large systems, small applications, internal systems, commercial software etc? (Note that I have not read the full version which may contain more information that I have missed) |
| Unfortunately no. |
| Not really. |
| I would suggest a simplification of the process with clear instructions on the steps to b |
| nope, it seems to have covered all the important bits |

## REFERENCES


[1] C. R. de Souza and D. F. Redmiles, "On the roles of apis in the coordination of collaborative software development," *Computer Supported Cooperative Work (CSCW)*, vol. 18, no. 5-6, p. 445, 2009.

[2] J. Stylos, S. Clarke, and B. Myers, "Comparing api design choices with usability studies: A case study and future directions," in *Proceedings of the 18th PPIG Workshop*, 2006.

[3] T. Aitamurto and S. C. Lewis, "Open innovation in digital journalism: Examining the impact of open apis at four news organizations," *new media & society*, vol. 15, no. 2, pp. 314–331, 2013.

[4] K. E. Hammouda Imed, Lindman Juho, "Api development." [Online]. Available: http://softwareday.lindholmen.se/en

[5] J. Kallinikos, A. Aaltonen, and A. Marton, "The ambivalent ontology of digital artifacts." *Mis Quarterly*, vol. 37, no. 2, pp. 357–370, 2013.

[6] R. H. Von Alan, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[7] D. Zowghi and C. Coulin, "Requirements elicitation: A survey of techniques, approaches, and tools," in *Engineering and managing software requirements*. Springer, 2005, pp. 19–46.

[8] L. Chen, M. A. Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE software*, vol. 30, no. 2, pp. 38–45, 2013.

[9] R. Wieringa, "Design science as nested problem solving," in *Proceedings of the 4th international conference on design science research in information systems and technology*. ACM, 2009, p. 8.

[10] V. K. Vaishnavi and W. Kuechler, *Design science research methods and patterns: innovating information and communication technology*. Crc Press, 2015.

[11] A. Collins, D. Joseph, and K. Bielaczyc, "Design research: Theoretical and methodological issues," *The Journal of the learning sciences*, vol. 13, no. 1, pp. 15–42, 2004.

[12] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical software engineering*, vol. 10, no. 3, pp. 311–341, 2005.

[13] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on software engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[14] K. Peffers, M. Rothenberger, T. Tuunanen, and R. Vaezi, "Design science research evaluation," in *International Conference on Design Science Research in Information Systems*. Springer, 2012, pp. 398–410.

[15] L. A. Goodman, "Snowball sampling," *The annals of mathematical statistics*, pp. 148–170, 1961.

[16] J. Grudin and J. Pruitt, "Personas, participatory design and product development: An infrastructure for engagement," in *PDC*, 2002, pp. 144–152.

[17] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[18] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.